SFU SIMON FRASER UNIVERSITY
BURNABY

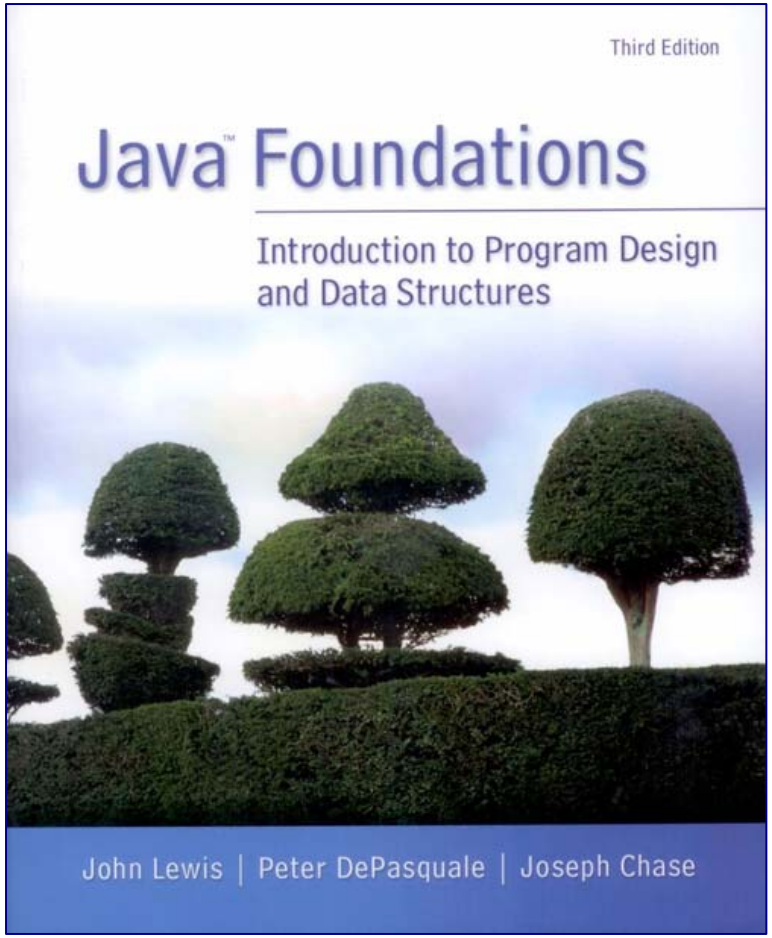
Introduction to Java

Chapters 1 and 2
The Java Language – Section 1.1
Data & Expressions – Sections 2.1 – 2.5

Instructor: Scott Kristjanson

CMPT 125/125

SFU Burnaby, Fall 2013



Scope

Introduce the Java programming language

- Program, Class, and Methods
- The Use of White Space and Comments
- Strings, Concatenation, and Escape Sequences
- Declaration and Use of Variables
- Java Primitive Data Types
- Syntax and Processing of Expressions
- Mechanisms for Data Conversion

Slides based on Java Foundations 3rd Edition, Lewis/DePasquale/Chase
And course material from Diana Cukierman, Lou Hafer, and Greg Baker

Java

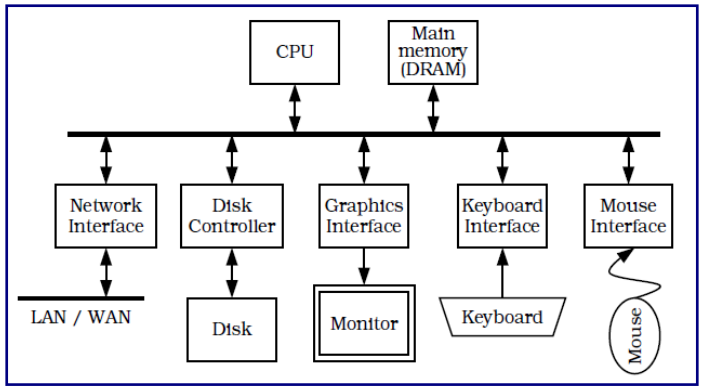
A computer is made up of hardware and software

- *hardware* – the physical, tangible pieces that support the computing effort
- *Program* – a series of instructions that the hardware executes

Programs are sometimes called *Applications*

Software

- consists of programs and the data those programs use
- Data includes files on disk such as pictures, templates, and databases
- Data can also be input from a user, the internet, or from devices



Slides based on Java Foundations 3rd Edition, Lewis/DePasquale/Chase
And course material from Diana Cukierman, Lou Hafer, and Greg Baker

Java

A *programming language* specifies the words and symbols that we can use to write a program

A programming language employs a set of rules that dictate how the words and symbols can be put together to form valid *program statements – this is called Syntax*

The Java programming language was created by Sun Microsystems, Inc.

It was introduced in 1995 and its popularity grew quickly, it is now the #1 most widely used programming language[2]

The Java Programming Language

In the Java programming language:

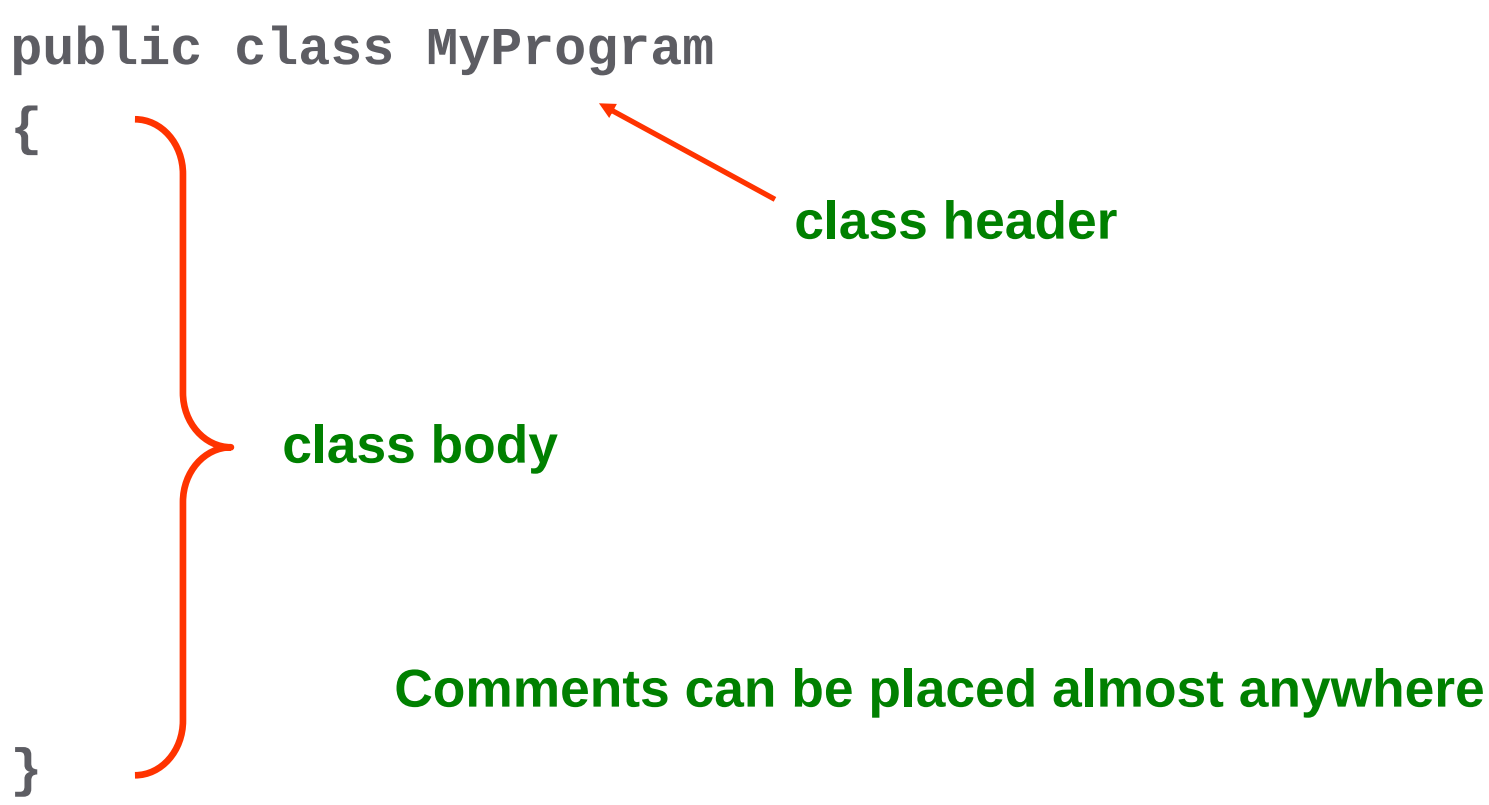
- a program is made up of one or more *classes*
- a class contains one or more *methods*
- a method contains program *statements*

These terms will be explored in detail throughout the course

A Java application always contains a method called **`main`**

Slides based on Java Foundations 3rd Edition, Lewis/DePasquale/Chase
And course material from Diana Cukierman, Lou Hafer, and Greg Baker

A Java Program



Slides based on Java Foundations 3rd Edition, Lewis/DePasquale/Chase
And course material from Diana Cukierman, Lou Hafer, and Greg Baker

A Java Program


```
//  comments about the class
public class MyProgram
{
   //  comments about the method
   public static void main(String[] args)
   {

   }

}
```

method body

method header

Slides based on Java Foundations 3rd Edition, Lewis/DePasquale/Chase
And course material from Diana Cukierman, Lou Hafer, and Greg Baker

Comments

Comments should be included to explain the purpose of the program and describe processing

Do not explain the obvious, explain the intent of the code at a higher level

They do not affect how a program works

Java comments can take three forms:

```
// this comment runs to the end of the line

/*  this comment runs to the terminating
    symbol, even across line breaks       */

/** this is a javadoc comment   */
```

A Very Simple Java Program


```
//********************************************************************
//  Lincoln.java       Java Foundations
//
//  Demonstrates the basic structure of a Java application.
//********************************************************************

public class Lincoln
{
   //-----------------------------------------------------------------
   // Prints a presidential quote.
   //-----------------------------------------------------------------
   public static void main(String[] args)
   {
      System.out.println("A quote by Abraham Lincoln:");

      System.out.println("Whatever you are, be a good one.");
   }
}
```

Comments about the class

class header

Comments about the method

method header

method body

class body

Output:

```
A quote by Abraham Lincoln:
Whatever you are, be a good one.
```


Slides based on Java Foundations 3rd Edition, Lewis/DePasquale/Chase
And course material from Diana Cukierman, Lou Hafer, and Greg Baker

Identifiers

Identifiers are the words a programmer uses in a program to name things

- can be made up of letters, digits, the underscore character (_), and the dollar sign
- cannot begin with a digit

Java is *case sensitive*

- `Total`, `total`, and `TOTAL` are different identifiers

By convention, programmers use different case styles for different types of identifiers, such as

- *title case* for class names - `Lincoln`
- *upper case* for constants - `MAXIMUM`

Identifiers

Sometimes we choose identifiers ourselves when writing a program (such as `Lincoln`)

Sometimes we are using another programmer's code, so we use the identifiers that he or she chose (such as `println`)

Often we use special identifiers called *reserved words* that already have a predefined meaning in the language

A reserved word cannot be used in any other way

Reserved Words

Java reserved words:

abstract	default	goto*	package	this
assert	do	if	private	throw
boolean	double	implements	protected	throws
break	else	import	public	transient
byte	enum	instanceof	return	true
case	extends	int	short	try
catch	false	interface	static	void
char	final	long	strictfp	volatile
class	finally	native	super	while
const*	float	new	switch	
continue	for	null	synchronized	

White Space

In Java:

- Spaces, blank lines, and tabs are called *white space*
- White space is used to separate words and symbols in a program
- A valid Java program can be formatted many ways
- Extra white space and indenting is ignored by the Java compiler
- Proper use of White Space is important – for ***people*** to understand it
- Programs should be formatted to enhance readability, using consistent indentation

A Poorly formatted version of Lincoln

Java may not care about format, but your reader does…
Use White Space to highlight program structure

Unclear White Space will lose marks for readability in your assignments!

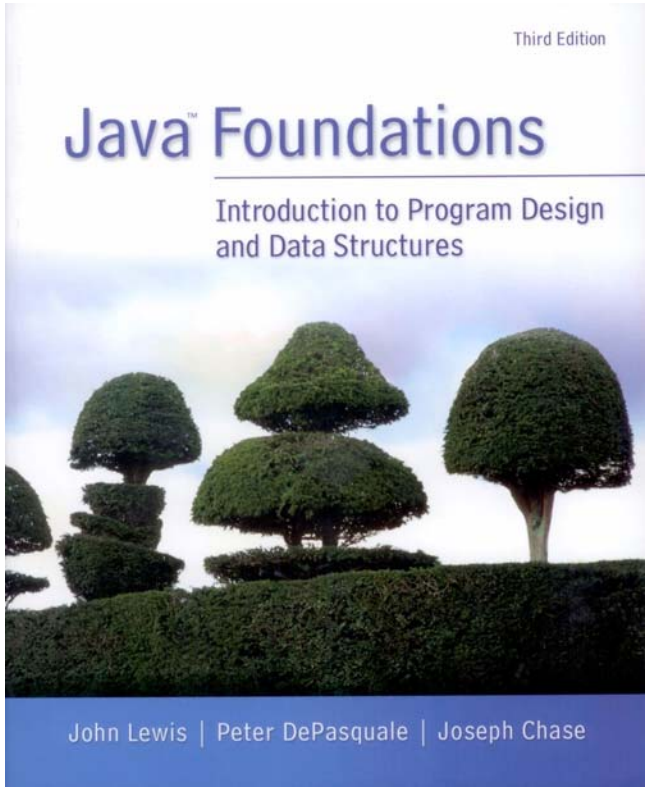
```
//********************************************************************
//  Lincoln2.java       Java Foundations
//
//  Demonstrates a poorly formatted, though valid, program.
//********************************************************************

public class Lincoln2{public static void main(String[]args){
System.out.println("A quote by Abraham Lincoln:");
System.out.println("Whatever you are, be a good one.");}}
```

A Horribly formatted version of Lincoln

This use of White Space is horribly unclear and could get you ZERO on an assignment!

```
//********************************************************************
//  Lincoln3.java       Java Foundations
//
//  Demonstrates another valid program that is poorly formatted.
//********************************************************************
          public       class
     Lincoln3
   {
                public
   static
        void
  main
        (
String
           []
    args                        )
  {
  System.out.println        (
"A quote by Abraham Lincoln:"          )
  ;        System.out.println
           (
       "Whatever you are, be a good one."
      )
  ;
}         }
```


Slides based on Java Foundations 3rd Edition, Lewis/DePasquale/Chase
And course material from Diana Cukierman, Lou Hafer, and Greg Baker



Chapter 2

Data & Expressions – Sections 2.1 – 2.5

Slides based on Java Foundations 3rd Edition, Lewis/DePasquale/Chase
And course material from Diana Cukierman, Lou Hafer, and Greg Baker

Scope

Character strings and concatenation
Escape sequences
Declaring and using variables
Java primitive types
Expressions
Data conversions

Slides based on Java Foundations 3rd Edition, Lewis/DePasquale/Chase
And course material from Diana Cukierman, Lou Hafer, and Greg Baker

Character Strings

A string of characters can be represented as a *string literal* by putting double quotes around it

Examples:

```
"This is a string literal."
"123 Main Street"
"X"
```

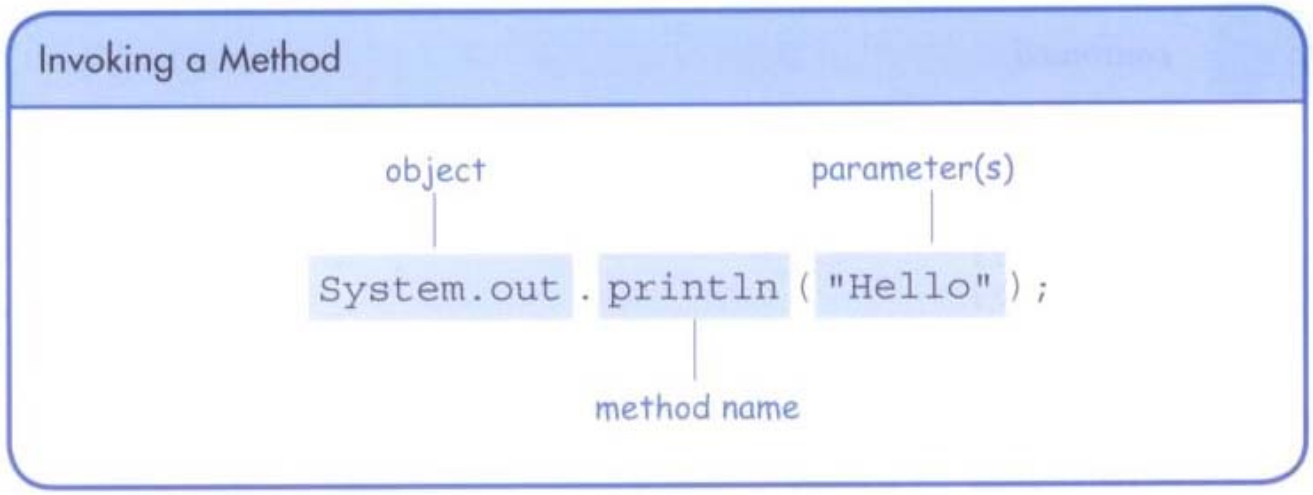
Every character string is an object in Java, defined by the **`String`** class

Every string literal represents a **`String`** object

The println Method

In the `Lincoln` program, we invoked the **`println`** method to print a character string

The **`System.out`** object represents a destination (the monitor) to which we can send output



Slides based on Java Foundations 3rd Edition, Lewis/DePasquale/Chase
And course material from Diana Cukierman, Lou Hafer, and Greg Baker

The print Method

The **System.out** object provides another service as well

The **print** method is similar to the **println** method, except that it does not advance to the next line

Therefore anything printed after a **print** statement will appear on the same line

```
public class Countdown
{
   //-----------------------------------------------------------------
   //  Prints two lines of output representing a rocket countdown.
   //-----------------------------------------------------------------
   public static void main(String[] args)
   {
      System.out.print("Three... ");
      System.out.print("Two... ");
      System.out.print("One... ");
      System.out.print("Zero... ");

      System.out.println("Liftoff!"); // appears on first output line

      System.out.println("Houston, we have a problem.");
   }
}
```

Output:

```
Three… Two… One… Zero… Liftoff!
Houston, we have a problem.
```


Slides based on Java Foundations 3rd Edition, Lewis/DePasquale/Chase
And course material from Diana Cukierman, Lou Hafer, and Greg Baker

String Concatenation

The *string concatenation operator* (+) is used to append one string to the end of another

```
"Peanut butter " + "and jelly"
```

It can also be used to append a number to a string

A string literal cannot be broken across two lines in a program

String Concatenation Example


```
//********************************************************************
//  Facts.java       Java Foundations
//  Demonstrates the use of the string concatenation operator and the
//  automatic conversion of an integer to a string.
//********************************************************************
public class Facts
{
   //-----------------------------------------------------------------
   //  Prints various facts.
   //-----------------------------------------------------------------
   public static void main(String[] args)
   {
      // Strings can be concatenated into one long string
      System.out.println("We present the following facts for your " + "extracurricular edification:");
      System.out.println();

      // A string can contain numeric digits
      System.out.println("Letters in the Hawaiian alphabet: 12");
      // A numeric value can be concatenated to a string
      System.out.println("Dialing code for Antarctica: " + 672);
      System.out.println("Year in which Leonardo da Vinci invented " + "the parachute: " + 1515);
      System.out.println("Speed of ketchup: " + 40 + " km per year");
   }
}
```

```
We present the following facts for your extracurricular edification:
Letters in the Hawaiian alphabet: 12
Dialing code for Antarctica: 672
Year in which Leonardo da Vinci invented the parachute: 1515
Speed of ketchup: 40 km per year
```

String Concatenation versus Addition

The + operator is also used for arithmetic addition

The function performed depends on the type of the operands

If both operands are strings, or if one is a string and one is a number, it performs string concatenation.

If both operands are numeric, it adds them.

The + operator is evaluated left to right, but parentheses can force the order

```
public class Addition
{
   //-----------------------------------------------------
   //  Concatenates and adds two numbers and prints the results.
   //-----------------------------------------------------
   public static void main(String[] args)
   {
      System.out.println("24 and 45 concatenated: " + 24 + 45);
      System.out.println("24 and 45 added: " + (24 + 45));
   }
}
```

```
24 and 45 concatenated: 2445
24 and 45 added: 69
```

Escape Sequences

What if we wanted to print a the quote character?

The following line would confuse the compiler because it would interpret the second quote as the end of the string

```
System.out.println("I said "Hello" to you.");
```

An *escape sequence* is a series of characters that represents a special character

An escape sequence begins with a backslash character (\)

```
System.out.println("I said \"Hello\" to you.");
```

```
I said "Hello" to you.
```


Slides based on Java Foundations 3rd Edition, Lewis/DePasquale/Chase
And course material from Diana Cukierman, Lou Hafer, and Greg Baker

Escape Sequences

Some Java escape sequences:

Escape Sequence	Meaning
\b	backspace
\t	tab
\n	newline
\r	carriage return
\"	double quote
\'	single quote
\\	backslash

Variables

A *variable* is a name for a location in memory

Before it can be used, a variable must be *declared* by specifying its name and the type of information that it will hold

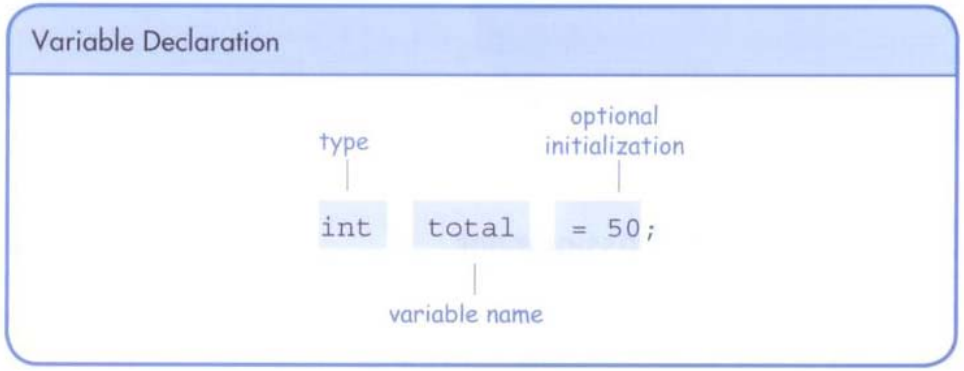
data type **variable name**

```
int total;

int count, temp, result;
```

Multiple variables can be created in one declaration

Variables

A variable can be given an initial value in the declaration



When a variable is used in a program, its current value is used

```
public class PianoKeys
{
   //-----------------------------------------------------
   //  Prints the number of keys on a piano.
   //-----------------------------------------------------
   public static void main(String[] args)
   {
      int keys = 88;
      System.out.println("A piano has " + keys + " keys.");
   }
}
```

```
A piano has 88 keys.
```


Slides based on Java Foundations 3rd Edition, Lewis/DePasquale/Chase
And course material from Diana Cukierman, Lou Hafer, and Greg Baker

Assignment

An *assignment statement* changes the value of a variable
The assignment operator is the = sign

```
total = 55;
```

The expression on the right is evaluated and the result is stored in the variable on the left

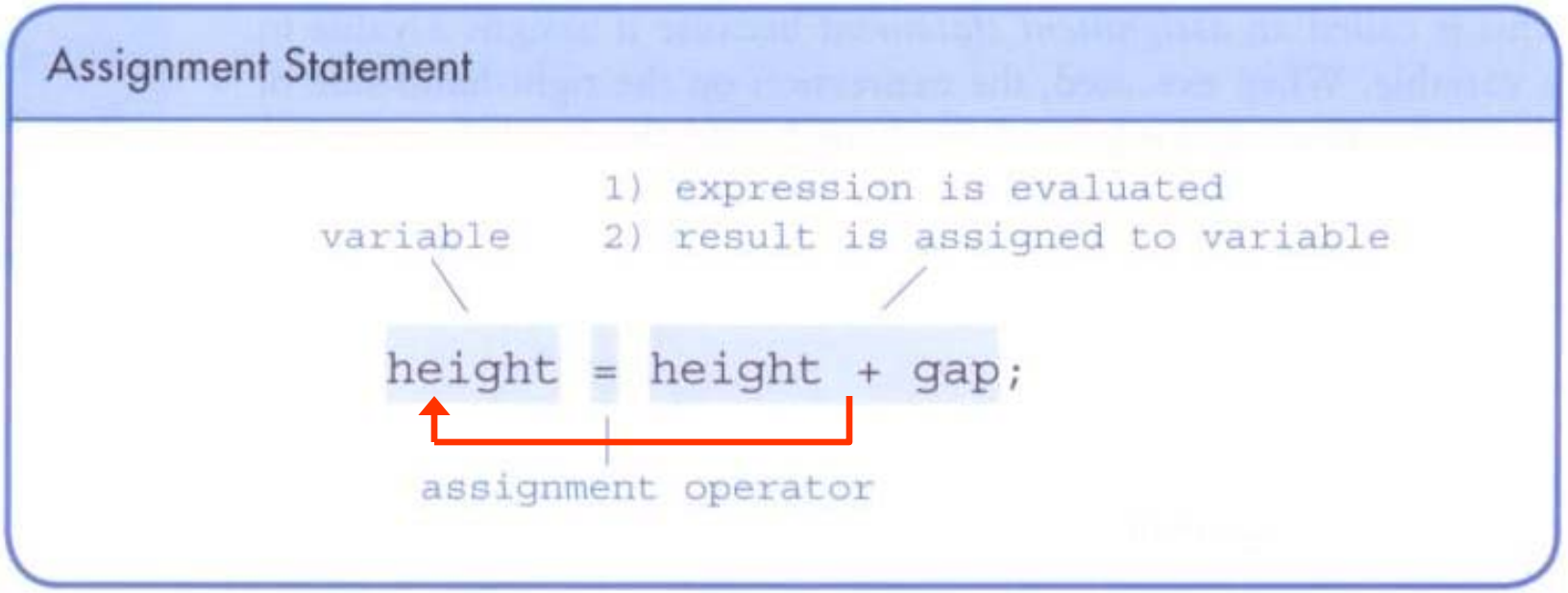
The value that was in `total` is overwritten

You can only assign a value to a variable that is consistent with the variable's declared type

Assignment

The right-hand side could be an expression

The expression on the right is completely evaluated and the result is stored in the variable identified on the left



Slides based on Java Foundations 3rd Edition, Lewis/DePasquale/Chase
And course material from Diana Cukierman, Lou Hafer, and Greg Baker

Constants

A *constant* is an identifier that is similar to a variable except that it holds the same value during its entire existence

As the name implies, it is constant, not variable

The compiler will issue an error if you try to change the value of a constant

In Java, we use the `final` modifier to declare a constant

```
final int MIN_HEIGHT = 69;
```

Constants

Constants are useful for three important reasons

- First, they give meaning to otherwise unclear literal values
  - For example, `MAX_LOAD` means more than the literal 250
- Second, they facilitate program maintenance
  - If a constant is used in multiple places, its value need only be updated in one place
- Third, they formally establish that a value should not change, avoiding inadvertent errors by other programmers

Primitive Data Types

There are eight primitive data types in Java

Four of them represent integers

- **`byte`**, **`short`**, **`int`**, **`long`**

Two of them represent floating point numbers

- **`float`**, **`double`**

One of them represents characters

- **`char`**

And one of them represents boolean values

- **`boolean`**

Numeric Types

The difference between the various numeric primitive types is their size, and therefore the values they can store:

Type	Storage	Min Value	Max Value
byte	8 bits	-128	127
short	16 bits	-32,768	32,767
int	32 bits	-2,147,483,648	2,147,483,647
long	64 bits	-9,223,372,036,854,775,808	9,223,372,036,854,775,807
float	32 bits	Approximately -3.4E+38 with 7 significant digits	Approximately 3.4E+38 with 7 significant digits
double	64 bits	Approximately -1.7E+308 with 15 significant digits	Approximately 1.7E+308 with 15 significant digits

Characters

A **`char`** variable stores a single character
Character literals are delimited by single quotes:

```
'a'   'X'   '7'   '$'   ','   '\n'
```

Example declarations

```
char topGrade = 'A';

char terminator = ';', separator = ' ';
```

Note the distinction between a primitive character variable, which holds only one character, and a **`String`** object, which can hold multiple characters

Character Sets

A *character set* is an ordered list of characters, with each character corresponding to a unique number

A **char** variable in Java can store any character from the *Unicode character set*

The Unicode character set uses sixteen bits per character
It is an international character set, containing symbols and characters from many world languages

Characters

The *ASCII character set* is older and smaller than Unicode

The ASCII characters are a subset of the Unicode character set, including:

uppercase letters	**A, B, C, …**
lowercase letters	**a, b, c, …**
punctuation	**period, semi-colon, …**
digits	**0, 1, 2, …**
special symbols	**&, |, \, …**
control characters	**carriage return, tab, ...**

Booleans

A **`boolean`** value represents a true or false condition

The reserved words `true` and `false` are the only valid values for a boolean type

```
boolean done = false;
```

A **`boolean`** variable can also be used to represent any two states, such as a light bulb being on or off

```
boolean Bulb_On = false;
```

Expressions

An *expression* is a combination of one or more operators and operands

Arithmetic expressions compute numeric results and make use of the arithmetic operators

- **Addition** **+**
- **Subtraction** **-**
- **Multiplication** *****
- **Division** **/**
- **Remainder** **%**

If either or both operands used by an arithmetic operator are floating point, then the result is a floating point

Division and Remainder

If both operands to the division operator (/) are integers, the result is an integer (the fractional part is discarded)

```
14 / 3      equals      4

8 / 12      equals      0
```

The remainder operator (%) returns the remainder after dividing the second operand into the first

```
14 % 3      equals      2
```

Operator Precedence

Operators can be combined into complex expressions

```
result  =  total + count / max - offset;
```

Operators have a well-defined precedence which determines the order in which they are evaluated

Multiplication, division, and remainder are evaluated prior to addition, subtraction, and string concatenation

Arithmetic operators with the same precedence are evaluated from left to right, but parentheses can be used to force the evaluation order

Operator Precedence

What is the order of evaluation in the following expressions?

```
a + b + c + d + e
```

```
a + b * c - d / e
```

```
a / (b + c) - d % e
```

```
a / (b * (c + (d - e)))
```

Operator Precedence

What is the order of evaluation in the following expressions?

```
a + b + c + d + e
  1   2   3   4
```

```
a + b * c - d / e
  3   1   4   2
```

```
a / (b + c) - d % e
  2    1    4   3
```

```
a / (b * (c + (d - e)))
  4    3    2    1
```

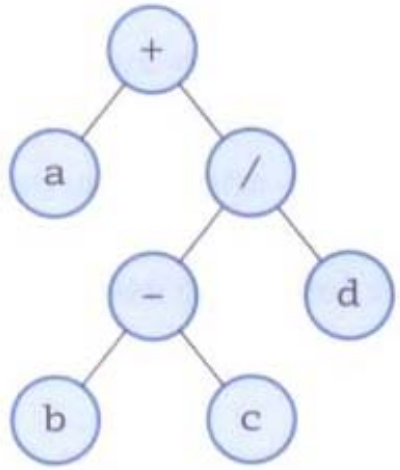

Slides based on Java Foundations 3rd Edition, Lewis/DePasquale/Chase
And course material from Diana Cukierman, Lou Hafer, and Greg Baker

Expression Trees

The evaluation of a particular expression can be shown using an *expression tree*

The operators lower in the tree have higher precedence for that expression

Evaluating
a + (b – c) / d



Slides based on Java Foundations 3rd Edition, Lewis/DePasquale/Chase
And course material from Diana Cukierman, Lou Hafer, and Greg Baker

Operator Precedence

Precedence among some Java operators:

Precedence Level	Operator	Operation	Associates
1	+	unary plus	R to L
	–	unary minus	
2	*	multiplication	L to R
	/	division	
	%	remainder	
3	+	addition	L to R
	–	subtraction	
	+	string concatenation	
4	=	assignment	R to L

Assignment Revisited

The assignment operator has a lower precedence than the arithmetic operators

First the expression on the right hand side of the = operator is evaluated

```
answer  =  sum / 4 + MAX * lowest;
           4       1   3     2
```

Then the result is stored in the variable on the left hand side

Slides based on Java Foundations 3rd Edition, Lewis/DePasquale/Chase
And course material from Diana Cukierman, Lou Hafer, and Greg Baker

Assignment Revisited

The right and left hand sides of an assignment statement can contain the same variable

First, one is added to the original value of count

```
count  =  count + 1;
```

Then the result is stored back into count (overwriting the original value)

Increment and Decrement Operators

The increment and decrement operators use only one operand

The *increment operator* (++) adds one to its operand
The *decrement operator* (--) subtracts one from its operand

The statement

```
count++;
```

is functionally equivalent to

```
count = count + 1;
```

Increment and Decrement Operators

The increment and decrement operators can be applied in two forms:

Postfix:

```
counter++; // Increment counter after returning its value
```

Prefix:

```
++counter; // Increment counter before returning its value
```

Because of their subtleties, the increment and decrement operators should be used with care until you have more experience with them.

When used as part of a larger expression, the two forms can have very different effects

What is the output from this code fragment?

```
int counter = 1;
System.out.println("counter = " + counter++ + ++counter);
```

```
counter = 13
```

Why 13? Does counter now equal 13?

Of course not! counter's final value is 3

Assignment Operators

Often we perform an operation on a variable, and then store the result back into that variable

Java provides *assignment operators* to simplify that process

For example, the statement

```
num += count;
```

is equivalent to

```
num = num + count;
```

Assignment Operators

There are many assignment operators in Java, including the following:

Operator	Example	Equivalent To
`+=`	`x += y`	`x = x + y`
`-=`	`x -= y`	`x = x - y`
`*=`	`x *= y`	`x = x * y`
`/=`	`x /= y`	`x = x / y`
`%=`	`x %= y`	`x = x % y`

Assignment Operators

The right hand side of an assignment operator can be a complex expression

The entire right-hand expression is evaluated first, then the result is combined with the original variable

Therefore

```
result /= (total-MIN) % num;
```

is equivalent to

```
result = result / ((total-MIN) % num);
```

Assignment Operators

The behavior of some assignment operators depends on the types of the operands

If the operands to the += operator are strings, the assignment operator performs string concatenation

The behavior of an assignment operator (+=) is always consistent with the behavior of the corresponding operator (+)

Data Conversions

Sometimes it is convenient to convert data from one type to another

For example, in a particular situation we may want to treat an integer as a floating point value

These conversions do not change the type of a variable or the value that's stored in it – they only convert a value as part of a computation

Data Conversions

Conversions must be handled carefully to avoid losing information

Widening conversions are safest because they tend to go from a small data type to a larger one (such as a `short` to an `int`)

Narrowing conversions can lose information because they tend to go from a large data type to a smaller one.

In Java, data conversions can occur in three ways

- assignment conversion
- promotion
- casting

Data Conversions

Widening Conversions

From	To
byte	short, int, long, float, or double
short	int, long, float, or double
char	int, long, float, or double
int	long, float, or double
long	float or double
float	double

Narrowing Conversions

From	To
byte	char
short	byte or char
char	byte or short
int	byte, short, or char
long	byte, short, char, or int
float	byte, short, char, int, or long
double	byte, short, char, int, long, or float

Assignment Conversion

Assignment conversion occurs when a value of one type is assigned to a variable of another

If **`money`** is a **`float`** variable and **`dollars`** is an **`int`** variable, the following assignment converts the value in **`dollars`** to a **`float`**

```
money = dollars
```

Only widening conversions can happen via assignment

Note that the value or type of **`dollars`** did not change

Promotion

Promotion happens automatically when operators in expressions convert their operands

For example, if **`sum`** is a **`float`** and **`count`** is an **`int`**, the value of **`count`** is converted to a floating point value to perform the following calculation

```
result = sum / count;
```

Casting

Casting is the most powerful, and dangerous, technique for conversion

Both widening and narrowing conversions can be accomplished by explicitly casting a value

To cast, the type is put in parentheses in front of the value being converted

For example, if **`total`** and **`count`** are integers, but we want a floating point result when dividing them, we can cast `total`

```
result = (float) total / count;
```

Key Things to take away:

- The print and println methods are two services provided by the System.out object
- In Java, the + operator is used both for addition and for string concatenation
- An escape character can be used to represent a character that would otherwise cause a compile error
- A variable is a name for a memory location used to hold a value of a particular data type
- Accessing data leaves them intact in memory, but an assignment statement overwrites old data
- One cannot assign a value of one type to a variable of an incompatible type
- Constants hold a particular value for the duration of their existence
- Java has two types of numeric values: integer and floating point. There are four integer data types and two floating point data types
- Java using 16-bit Unicode character set to represent character data
- Expressions are combinations of operators and operands used to perform a calculation
- The type of result produced by arithmetic division depends on the types of the operands
- Java follows a well-defined set of precedence rules that governs the order in which operators will be evaluated in an expression
- Narrowing conversions should be avoided because they can lose information

References:

1. J. Lewis, P. DePasquale, and J. Chase., *Java Foundations: Introduction to Program Design & Data Structures*. Addison-Wesley, Boston, Massachusetts, 3rd edition, 2014, ISBN 978-0-13-337046-1

2. *Top 10 Most Popular Programming Languages* website, http://www.english4it.com/reading/40

3. L. Hafer, *Computing Science 125/126 Course Notes, Summer 2013*

4. *D. Cukierman, Computing Science 125/126 Course Notes, Spring 2012*

5. G. Baker, *The Computing Science 120 Study Guide: Introduction to Computing Science and Programming I*, Fall 2010 Edition, http://www2.cs.sfu.ca/CourseCentral/120/ggbaker/guide/guide

Time for Questions